Q1. Explain the difference between greedy and non-greedy syntax with visual terms in as few words as possible. What is the bare minimum effort required to transform a greedy pattern into a non-greedy one? What characters or characters can you introduce or change?

Greedy syntax matches the longest possible string, while non-greedy syntax matches the shortest possible string. To transform a greedy pattern into a non-greedy one, you can introduce a question mark "?" after the quantifier (e.g., \*? or +?) or change the quantifier itself to its non-greedy counterpart (e.g., \* to \*? or + to +?).

Q2. When exactly does greedy versus non-greedy make a difference?  What if you're looking for a non-greedy match but the only one available is greedy?

The distinction between greedy and non-greedy matching matters when there are multiple potential matches within the input string. Greedy matching captures the longest possible substring, while non-greedy matching captures the shortest. If a non-greedy match is desired but only a greedy match is available, it may be necessary to modify the pattern or input to achieve the desired non-greedy behavior.

Q3. In a simple match of a string, which looks only for one match and does not do any replacement, is the use of a nontagged group likely to make any practical difference?

In a simple match of a string without any replacement, the use of a non-tagged group (a group without capturing parentheses) does not typically make a practical difference. Non-tagged groups are primarily used for grouping and applying quantifiers, but they do not capture the matched substring separately. Therefore, in a simple match scenario, the choice between tagged and non-tagged groups does not have a significant impact.

Q4. Describe a scenario in which using a nontagged category would have a significant impact on the program's outcomes.

A scenario where using a non-tagged category (a group without capturing parentheses) would have a significant impact is when performing advanced string manipulation or replacement operations. By using tagged groups, the captured substrings can be referenced and utilized in the replacement pattern. Non-tagged groups would be useful when you want to group expressions for quantification purposes but do not need to reference or manipulate the captured substrings individually.

Q5. Unlike a normal regex pattern, a look-ahead condition does not consume the characters it examines. Describe a situation in which this could make a difference in the results of your programme.

The look-ahead condition not consuming characters it examines can make a difference in scenarios where there are overlapping patterns. For example, suppose you want to match all occurrences of the word "book" followed by "shelf" in a text. Using a look-ahead condition allows you to find overlapping matches without consuming characters, ensuring that subsequent matches are not skipped due to overlapping substrings.

Q6. In standard expressions, what is the difference between positive look-ahead and negative look-ahead?

In standard regular expressions, positive look-ahead and negative look-ahead are both types of look-ahead assertions.

Positive look-ahead (syntax: `(?=...)`) asserts that the pattern inside the lookahead must match, but it does not consume any characters in the process. It ensures that the pattern is followed by another pattern.

Negative look-ahead (syntax: `(?!...)`) asserts that the pattern inside the lookahead must not match. It also does not consume any characters and ensures that the pattern is not followed by another specific pattern.

Q7. What is the benefit of referring to groups by name rather than by number in a standard expression?

Referring to groups by name instead of by number in a standard regular expression brings several benefits:

1. Improved Readability: Using named groups makes the regular expression more readable and self-explanatory, as the purpose of each group is explicitly labeled with a meaningful name.

2. Code Maintainability: If the regular expression needs to be modified or extended, using named groups simplifies the process by allowing changes to be made based on the group names rather than relying on numerical indices.

3. Self-Documenting Patterns: Named groups provide documentation within the regular expression itself, making it easier for developers to understand the purpose and intention of each group.

4. Accessible Captured Substrings: Named groups allow easy access to the captured substrings by their assigned names, which can be beneficial when extracting and working with specific parts of the matched text.

Overall, using named groups in regular expressions enhances code clarity, maintainability, and accessibility of captured substrings, contributing to more robust and readable code.

Q8. Can you identify repeated items within a target string using named groups, as in "The cow jumped over the moon"?

Yes, repeated items within a target string can be identified using named groups in regular expressions. For example, to identify repeated words in the sentence "The cow jumped over the moon," you can use the following pattern: `\b(?<word>\w+)\b\s+(\k<word>)\b`. This pattern captures a word with the named group "word" and then matches the same word again using `\k<word>`.

Q9. When parsing a string, what is at least one thing that the Scanner interface does for you that the re.findall feature does not?

One thing that the Scanner interface provides, which the `re.findall` feature does not, is the ability to scan and extract tokenized values from a string. The Scanner interface allows you to specify delimiters or patterns to identify and extract individual tokens. In contrast, `re.findall` focuses on pattern matching and returns all non-overlapping matches of a given pattern, without the concept of tokenization.

Q10. Does a scanner object have to be named scanner?

No, a Scanner object does not have to be named "scanner." The name given to a Scanner object is arbitrary and can be chosen according to the programmer's preference. It is common practice to choose a descriptive name that reflects the purpose of the Scanner object or the data it is processing, but the name itself does not affect the functionality of the object.